cachelab

> *Author: 杨佳宇轩*

模拟高速缓存

> `csim.c` & `csim-clock.c` & `csim-LFU.c` & `csim-FIFO.c`

`csim.c` - LRU - 最近最少使用

```c
#include "cachelab.h"
#include <getopt.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define rep(i, a, b) for(int i = a; i <= b; i++)
#define per(i, a, b) for(int i = a; i >= b; i--)

int h, v, s, b, E, t, S;

int hit_cnt, miss_cnt, alter_cnt;

char path[1005];

typedef struct {
    int valid;
    unsigned long long tag;
    int stamp;
}Cache_line, *Cache_gro, **Cache;

Cache cache = NULL;

void init_cache() {
    cache = (Cache)malloc(sizeof(Cache_gro) * S);
    rep(i, 0, S - 1) {
        cache[i] = (Cache_gro)malloc(sizeof(Cache_line) * E);
        rep(j, 0, E - 1) {
            cache[i][j].valid = 0;
            cache[i][j].tag = -1;
            cache[i][j].stamp = -1;
        }
    }
}

void update_stamp() {
    rep(i, 0, S - 1) rep(j, 0, E - 1)
        if(cache[i][j].valid == 1)
            cache[i][j].stamp++;
}

void update(unsigned long long address) {
    int setIndex = (address >> b) & ((-1ull) >> (64 - s));
    unsigned long long tag_add = address >> (b + s);

    // whether hit
    rep(i, 0, E - 1) {
        if(cache[setIndex][i].tag == tag_add) {
            cache[setIndex][i].stamp = 0;
            hit_cnt++;
            if(v) printf(" hit");
            return;
        }
    }
    miss_cnt++;
    if(v) printf(" miss");

    // whether empty line
    rep(i, 0, E - 1) {
        if(cache[setIndex][i].valid == 0) {
            cache[setIndex][i].tag = tag_add;
            cache[setIndex][i].valid = 1;
            cache[setIndex][i].stamp = 0;
            return;
        }
    }
    alter_cnt++;
    if(v) printf(" eviction");

    // alter
    int mx = -1, id = -1;
    rep(i, 0, E - 1) {
        if(cache[setIndex][i].stamp > mx) {
            mx = cache[setIndex][i].stamp;
            id = i;
        }
    }
    cache[setIndex][id].stamp = 0;
    cache[setIndex][id].tag = tag_add;
}

void process() {
    FILE *f = fopen(path, "r");

    char opt;
    unsigned long long address;
    int size;
    while(fscanf(f, " %c %llx,%d", &opt, &address, &size)
!= EOF) {
        if(v) printf("%c %llx,%d", opt, address, size);
        switch(opt) {
            case 'I': break;
            case 'L': update(address); break;
            case 'M': update(address); update(address);
break;
            case 'S': update(address); break;
        }
        if(v) printf("\n");
        update_stamp();
    }

    fclose(f);
}

void free_cache() {
    rep(i, 0, S - 1)
        free(cache[i]);
    free(cache);
}

int main(int argc, char* argv[]) {

    int opt;
    while(-1 != (opt = (getopt(argc, argv,
"hvs:E:b:t:")))) {
        switch(opt) {
            case 'h': h = 1; break; // pass
            case 'v': v = 1; break;
            case 's': s = atoi(optarg); S = 1 << s; break;
            case 'E': E = atoi(optarg); break;
            case 'b': b = atoi(optarg); break;
            case 't': strcpy(path, optarg); break;
            default: break;
        }
    }

    init_cache();
    process();
    free_cache();
    printSummary(hit_cnt, miss_cnt, alter_cnt);

    return 0;
}
```

`csim-clock.c` - NRU - 时钟

```
#include "cachelab.h"
```

```
#include <getopt.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define rep(i, a, b) for(int i = a; i <= b; i++)
#define per(i, a, b) for(int i = a; i >= b; i--)

int h, v, s, b, E, t, S;

int hit_cnt, miss_cnt, alter_cnt;

char path[1005];

typedef struct {
    int valid;
    unsigned long long tag;
    int vis;
}Cache_line, *Cache_gro, **Cache;
int clock_cnt[1500];

Cache cache = NULL;

void init_cache() {
    cache = (Cache)malloc(sizeof(Cache_gro) * S);
    rep(i, 0, S - 1) {
        cache[i] = (Cache_gro)malloc(sizeof(Cache_line) *
E);
        rep(j, 0, E - 1) {
            cache[i][j].valid = 0;
            cache[i][j].tag = -1;
            cache[i][j].vis = 0;
        }
    }
}

void update(unsigned long long address) {
    int setIndex = (address >> b) & ((-1ull) >> (64 - s));
    unsigned long long tag_add = address >> (b + s);

    // whether hit
    rep(i, 0, E - 1) {
        if(cache[setIndex][i].tag == tag_add) {
            cache[setIndex][i].vis = 1;
            hit_cnt++;
            if(v) printf(" hit");
            return;
        }
    }
    miss_cnt++;
    if(v) printf(" miss");
    // whether empty line
    rep(i, 0, E - 1) {
        if(cache[setIndex][i].valid == 0) {
            cache[setIndex][i].tag = tag_add;
            cache[setIndex][i].valid = 1;
            cache[setIndex][i].vis = 1;

            clock_cnt[setIndex] = (clock_cnt[setIndex] + 1) % E;
            return;
        }
    }
    alter_cnt++;
    if(v) printf(" eviction");

    // alter
    while(1) {
        if(cache[setIndex][clock_cnt[setIndex]].vis == 1) {
            cache[setIndex][clock_cnt[setIndex]].vis = 0;
            clock_cnt[setIndex] = (clock_cnt[setIndex] + 1) % E;
        } else if(cache[setIndex][clock_cnt[setIndex]].vis == 0) {
            cache[setIndex][clock_cnt[setIndex]].tag = tag_add;
            cache[setIndex][clock_cnt[setIndex]].vis = 1;
            clock_cnt[setIndex] = (clock_cnt[setIndex] + 1) % E;
            break;
        }
    }
}

void process() {
    FILE *f = fopen(path, "r");

    char opt;
    unsigned long long address;
    int size;

    while(fscanf(f, " %c %llx,%d", &opt, &address, &size) != EOF) {
        if(v) printf("%c %llx,%d", opt, address, size);
        switch(opt) {
            case 'I': break;
            case 'L': update(address); break;
            case 'M': update(address); update(address); break;
            case 'S': update(address); break;
        }
        if(v) printf("\n");
    }

    fclose(f);
}

void free_cache() {
    rep(i, 0, S - 1)
        free(cache[i]);
    free(cache);
}

int main(int argc, char* argv[]) {

    int opt;
    while(-1 != (opt = (getopt(argc, argv,
"hvs:E:b:t:")))) {
        switch(opt) {
            case 'h': h = 1; break; // pass
            case 'v': v = 1; break;
            case 's': s = atoi(optarg); S = 1 << s; break;
            case 'E': E = atoi(optarg); break;
            case 'b': b = atoi(optarg); break;
            case 't': strcpy(path, optarg); break;
            default: break;
        }
    }

    init_cache();
    process();
    free_cache();
    printSummary(hit_cnt, miss_cnt, alter_cnt);

    return 0;
}
```

csim-FLU.c - LFU - 最少使用频率

```
#include "cachelab.h"
#include <getopt.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define rep(i, a, b) for(int i = a; i <= b; i++)
#define per(i, a, b) for(int i = a; i >= b; i--)

int h, v, s, b, E, t, S;

int hit_cnt, miss_cnt, alter_cnt;
```

```c
char path[1005];

typedef struct {
    int valid;
    unsigned long long tag;
    int stamp;
}Cache_line, *Cache_gro, **Cache;

Cache cache = NULL;

void init_cache() {
    cache = (Cache)malloc(sizeof(Cache_gro) * S);
    rep(i, 0, S - 1) {
        cache[i] = (Cache_gro)malloc(sizeof(Cache_line) *
E);
        rep(j, 0, E - 1) {
            cache[i][j].valid = 0;
            cache[i][j].tag = -1;
            cache[i][j].stamp = -1;
        }
    }
}

void update(unsigned long long address) {
    int setIndex = (address >> b) & ((-1ull) >> (64 - s));
    unsigned long long tag_add = address >> (b + s);

    // whether hit
    rep(i, 0, E - 1) {
        if(cache[setIndex][i].tag == tag_add) {
            cache[setIndex][i].stamp++;
            hit_cnt++;
            if(v) printf(" hit");
            return;
        }
    }
    miss_cnt++;
    if(v) printf(" miss");

    // whether empty line
    rep(i, 0, E - 1) {
        if(cache[setIndex][i].valid == 0) {
            cache[setIndex][i].tag = tag_add;
            cache[setIndex][i].valid = 1;
            cache[setIndex][i].stamp = 1;
            return;
        }
    }
    alter_cnt++;
    if(v) printf(" eviction");

    // alter
    int mn = 0x7fffffff, id = -1;
    rep(i, 0, E - 1) {
        if(cache[setIndex][i].stamp < mn) {
            mn = cache[setIndex][i].stamp;
            id = i;
        }
    }
    cache[setIndex][id].stamp = 1;
    cache[setIndex][id].tag = tag_add;
}

void process() {
    FILE *f = fopen(path, "r");

    char opt;
    unsigned long long address;
    int size;

    while(fscanf(f, " %c %llx,%d", &opt, &address, &size)
!= EOF) {
        if(v) printf("%c %llx,%d", opt, address, size);
        switch(opt) {
            case 'I': break;
            case 'L': update(address); break;
            case 'M': update(address); update(address);
break;
            case 'S': update(address); break;
        }
        if(v) printf("\n");
    }

    fclose(f);
}

void free_cache() {
    rep(i, 0, S - 1)
        free(cache[i]);
    free(cache);
}

int main(int argc, char* argv[]) {

    int opt;
    while(-1 != (opt = (getopt(argc, argv,
"hvs:E:b:t:")))) {
        switch(opt) {
            case 'h': h = 1; break; // pass
            case 'v': v = 1; break;
            case 's': s = atoi(optarg); S = 1 << s; break;
            case 'E': E = atoi(optarg); break;
            case 'b': b = atoi(optarg); break;
            case 't': strcpy(path, optarg); break;
            default: break;
        }
    }

    init_cache();
    process();
    free_cache();
    printSummary(hit_cnt, miss_cnt, alter_cnt);

    return 0;
}
```

`csim-FIFO.c` - FIFO - 先进先出

```
#include "cachelab.h"
#include <getopt.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define rep(i, a, b) for(int i = a; i <= b; i++)
#define per(i, a, b) for(int i = a; i >= b; i--)

int h, v, s, b, E, t, S;

int hit_cnt, miss_cnt, alter_cnt;

char path[1005];

typedef struct {
    int valid;
    unsigned long long tag;
}Cache_line, *Cache_gro, **Cache;

int Q[1500][20], head[1500], tail[1500];

Cache cache = NULL;

void init_cache() {
    rep(i, 0, S - 1) tail[i] = -1;
    cache = (Cache)malloc(sizeof(Cache_gro) * S);
    rep(i, 0, S - 1) {
        cache[i] = (Cache_gro)malloc(sizeof(Cache_line) *
E);
        rep(j, 0, E - 1) {
            cache[i][j].valid = 0;
            cache[i][j].tag = -1;
        }
    }
}
```

```
void update(unsigned long long address) {
    int setIndex = (address >> b) & ((-1ull) >> (64 - s));
    unsigned long long tag_add = address >> (b + s);

    // whether hit
    rep(i, 0, E - 1) {
        if(cache[setIndex][i].tag == tag_add) {
            hit_cnt++;
            if(v) printf(" hit");
            return;
        }
    }
    miss_cnt++;
    if(v) printf(" miss");

    // whether empty line
    rep(i, 0, E - 1) {
        if(cache[setIndex][i].valid == 0) {
            tail[setIndex] = (tail[setIndex] + 1) % E;
            Q[setIndex][tail[setIndex]] = i;
            cache[setIndex][i].tag = tag_add;
            cache[setIndex][i].valid = 1;
            return;
        }
    }
    alter_cnt++;
    if(v) printf(" eviction");

    // alter
    tail[setIndex] = (tail[setIndex] + 1) % E;
    cache[setIndex][Q[setIndex][tail[setIndex]]].tag =
tag_add;
}

void process() {
    FILE *f = fopen(path, "r");

    char opt;
    unsigned long long address;
    int size;

    while(fscanf(f, " %c %llx,%d", &opt, &address, &size)
!= EOF) {
        if(v) printf("%c %llx,%d", opt, address, size);
        switch(opt) {
            case 'I': break;
            case 'L': update(address); break;
            case 'M': update(address); update(address);
break;
            case 'S': update(address); break;
        }
        if(v) printf("\n");
    }

    fclose(f);
}

void free_cache() {
    rep(i, 0, S - 1)
        free(cache[i]);
    free(cache);
}

int main(int argc, char* argv[]) {

    int opt;
    while(-1 != (opt = (getopt(argc, argv,
"hvs:E:b:t:")))) {
        switch(opt) {
            case 'h': h = 1; break; // pass
            case 'v': v = 1; break;
            case 's': s = atoi(optarg); S = 1 << s; break;
            case 'E': E = atoi(optarg); break;
            case 'b': b = atoi(optarg); break;
            case 't': strcpy(path, optarg); break;
            default: break;
        }
    }

    init_cache();
    process();
    free_cache();
    printSummary(hit_cnt, miss_cnt, alter_cnt);

    return 0;
}
```

64x64 理论最优解 1024

```
// with the original operation (without blocks-shifting
and lazy-transposing), 1176 = 35 * 8 + 16 * 56
// with block-shifting and lazy-transposing, it reaches
theoratical limit: 1024 = 16 * 64
void transpose_64(int M, int N, int A[N][M], int B[M][N]){
    // loop indices
    int i, j, ii, jj;
    // temporary variables
    int a0, a1, a2, a3, a4, a5, a6, a7;
    // main loop: ii, jj for each block of size 8x8

    for (jj = 0; jj < N; jj += 8){
        // process diagonal blocks first
        // ii: j-index of target block (block-shifting)
        // more specifically, use the upper half of [jj,
ii] to transpose [jj, jj] block
        // the target block is the one that will be used
immediately after the diagonal processing
        if (jj == 0) ii = 8; else ii = 0;

        // move the lower 4x8 blocks from A to B, with
block-shifting to the target block
        for (i = jj; i < jj + 4; ++i){
            a0 = A[i+4][jj+0];
            a1 = A[i+4][jj+1];
            a2 = A[i+4][jj+2];
            a3 = A[i+4][jj+3];
            a4 = A[i+4][jj+4];
            a5 = A[i+4][jj+5];
            a6 = A[i+4][jj+6];
            a7 = A[i+4][jj+7];

            B[i][ii+0] = a0;
            B[i][ii+1] = a1;
            B[i][ii+2] = a2;
            B[i][ii+3] = a3;
            B[i][ii+4] = a4;
            B[i][ii+5] = a5;
            B[i][ii+6] = a6;
            B[i][ii+7] = a7;
        }

        // taking transpose of lower-left and lower-right
4x4 within themselves respectively
        for (i = 0; i < 4; ++ i){
            for (j = i + 1; j < 4; ++j){
                a0 = B[jj+i][ii+j];
                B[jj+i][ii+j] = B[jj+j][ii+i];
                B[jj+j][ii+i] = a0;

                a0 = B[jj+i][ii+j+4];
                B[jj+i][ii+j+4] = B[jj+j][ii+i+4];
                B[jj+j][ii+i+4] = a0;
            }
        }

        // moving the upper 4x8 blocks from A to B
        for (i = jj; i < jj + 4; ++i){
            a0 = A[i][jj+0];
            a1 = A[i][jj+1];
            a2 = A[i][jj+2];
            a3 = A[i][jj+3];
            a4 = A[i][jj+4];
            a5 = A[i][jj+5];
            a6 = A[i][jj+6];
            a7 = A[i][jj+7];

            B[i][jj+0] = a0;
            B[i][jj+1] = a1;
            B[i][jj+2] = a2;
            B[i][jj+3] = a3;
            B[i][jj+4] = a4;
            B[i][jj+5] = a5;
            B[i][jj+6] = a6;
            B[i][jj+7] = a7;
        }

        // taking transpose of upper-left and upper-right
4x4 within themselves respectively
        for (i = jj; i < jj + 4; ++i){
            for (j = i + 1; j < jj + 4; ++j){
                a0 = B[i][j];
                B[i][j] = B[j][i];
                B[j][i] = a0;

                a0 = B[i][j+4];
                B[i][j+4] = B[j][i+4];
                B[j][i+4] = a0;
            }
        }

        // swaping the lower-left and upper-right
        for (i = 0; i < 4; ++ i){
            a0 = B[jj+i][jj+4];
            a1 = B[jj+i][jj+5];
            a2 = B[jj+i][jj+6];
            a3 = B[jj+i][jj+7];

            B[jj+i][jj+4] = B[jj+i][ii+0];
            B[jj+i][jj+5] = B[jj+i][ii+1];
            B[jj+i][jj+6] = B[jj+i][ii+2];
            B[jj+i][jj+7] = B[jj+i][ii+3];

            B[jj+i][ii+0] = a0;
            B[jj+i][ii+1] = a1;
            B[jj+i][ii+2] = a2;
            B[jj+i][ii+3] = a3;

        }

        // filling the original lower 4x8 from the block-
shifting block
        for (i = 0; i < 4; ++ i){
            B[jj+i+4][jj+0] = B[jj+i][ii+0];
            B[jj+i+4][jj+1] = B[jj+i][ii+1];
            B[jj+i+4][jj+2] = B[jj+i][ii+2];
```

```
      B[jj+i+4][jj+3] = B[jj+i][ii+3];
      B[jj+i+4][jj+4] = B[jj+i][ii+4];
      B[jj+i+4][jj+5] = B[jj+i][ii+5];
      B[jj+i+4][jj+6] = B[jj+i][ii+6];
      B[jj+i+4][jj+7] = B[jj+i][ii+7];
    }

    // processing off-diagonal blocks
    for (ii = 0; ii < M; ii += 8){
      if (ii == jj){
        // skip diagonal blocks
        continue;
      }else{
        // taking transpose of upper-left 4x4 and
upper-right 4x4 within themselves respectively
        for (i = ii; i < ii + 4; ++i){
          a0 = A[i][jj+0];
          a1 = A[i][jj+1];
          a2 = A[i][jj+2];
          a3 = A[i][jj+3];
          a4 = A[i][jj+4];
          a5 = A[i][jj+5];
          a6 = A[i][jj+6];
          a7 = A[i][jj+7];

          B[jj+0][i] = a0;
          B[jj+1][i] = a1;
          B[jj+2][i] = a2;
          B[jj+3][i] = a3;
          B[jj+0][i+4] = a4;
          B[jj+1][i+4] = a5;
          B[jj+2][i+4] = a6;
          B[jj+3][i+4] = a7;
        }

        // taking transpose of lower-left 4x4 and
store to upper-right 4x4, and move upper-right 4x4 to
lower-left 4x4
        for (j = jj; j < jj + 4; ++j){
          a0 = A[ii+4][j];
          a1 = A[ii+5][j];
          a2 = A[ii+6][j];
          a3 = A[ii+7][j];

          a4 = B[j][ii+4];
          a5 = B[j][ii+5];
          a6 = B[j][ii+6];
          a7 = B[j][ii+7];

          B[j][ii+4] = a0;
          B[j][ii+5] = a1;
          B[j][ii+6] = a2;
```

```
                    B[j][ii+7] = a3;

                    B[j+4][ii+0] = a4;
                    B[j+4][ii+1] = a5;
                    B[j+4][ii+2] = a6;
                    B[j+4][ii+3] = a7;
                }

                // taking transpose of lower-right 4x4
                for (i = ii + 4; i < ii + 8; ++i){
                    a0 = A[i][jj+4];
                    a1 = A[i][jj+5];
                    a2 = A[i][jj+6];
                    a3 = A[i][jj+7];

                    B[jj+4][i] = a0;
                    B[jj+5][i] = a1;
                    B[jj+6][i] = a2;
                    B[jj+7][i] = a3;
                }
            }
        }
    }
}
```

矩阵转置

> *直接转置 - 临时变量存后转置 - 复制后再转置 - 乱分块直接转置 - 右上暂存法*

`trans.c`

```
/*
 * trans.c - Matrix transpose B = A^T
 *
 * Your transpose function must have a prototype of the form:
 * void trans(int M, int N, int A[N][M], int B[M][N]);
 *
 * The transpose function is evaluated by counting the number of misses
 * on a 512B direct mapped cache with a block size of 32 bytes.
 */
#include <stdio.h>
#include "cachelab.h"

/*
 * transpose_submit - This is the solution transpose function that you
 *      will be graded on for Part B of the assignment. Do not change
 *      the description string "Transpose submission", as the driver
 *      searches for that string to identify the transpose function to
 *      be graded.
 */

void transpose_submit(int M, int N, int A[N][M], int B[M][N])
{
    /*
     * Please define your local variables here.
     * You are allowed to define at most 12 local variables.
    */
    int i, j, x, y;
    int e0, e1, e2, e3, e4, e5, e6, e7;

    if(M == 16) { //16x16 matrix
       for(i = 0; i < 16; i += 8) {
            for(j = 0; j < 16; j += 8) {
                // diag: 11 * 4
                for(x = i; x < i + 4; x++) {
                    e0 = A[x][j];
                    e1 = A[x][j + 1];
                    e2 = A[x][j + 2];
                    e3 = A[x][j + 3];
                    e4 = A[x][j + 4];
                    e5 = A[x][j + 5];
                    e6 = A[x][j + 6];
                    e7 = A[x][j + 7];

                    B[j][x] = e0;
                    B[j + 1][x] = e1;
                    B[j + 2][x] = e2;
                    B[j + 3][x] = e3;
                    B[j][x + 4] = e4;
                    B[j + 1][x + 4] = e5;
                    B[j + 2][x + 4] = e6;
                    B[j + 3][x + 4] = e7;
                }

                // diag: 15 * 4
                for(x = j; x < j + 4; x++) {
                    e0 = A[i + 4][x];
                    e1 = A[i + 5][x];
                    e2 = A[i + 6][x];
                    e3 = A[i + 7][x];
                    e4 = B[x][i + 4];
                    e5 = B[x][i + 5];
                    e6 = B[x][i + 6];
                    e7 = B[x][i + 7];

                    B[x][i + 4] = e0;
                    B[x][i + 5] = e1;
                    B[x][i + 6] = e2;
                    B[x][i + 7] = e3;
                    B[x + 4][i] = e4;
                    B[x + 4][i + 1] = e5;
                    B[x + 4][i + 2] = e6;
                    B[x + 4][i + 3] = e7;
                }

                // diag: 9 * 4
                for(x = i + 4; x < i + 8; x++) {
                    e0 = A[x][j + 4];
                    e1 = A[x][j + 5];
                    e2 = A[x][j + 6];
                    e3 = A[x][j + 7];
                    B[j + 4][x] = e0;
                    B[j + 5][x] = e1;
                    B[j + 6][x] = e2;
                    B[j + 7][x] = e3;
                }
            }
        }
        /**
         * for 16x16, 64(+3) misses
         */
        // for(i = 0; i < 16; i += 8) {
        //     for(j = 0; j < 16; j += 8) {
        //         if(i == j) {
        //             e0 = A[i][j];
        //             e1 = A[i][j + 1];
        //             e2 = A[i][j + 2];
        //             e3 = A[i][j + 3];
        //             e4 = A[i][j + 4];
        //             e5 = A[i][j + 5];
        //             e6 = A[i][j + 6];
        //             e7 = A[i][j + 7];

        //             for(x = i + 1; x < i + 8; x++) {
        //                 B[x - 1][j] = e0;
        //                 B[x - 1][j + 1] = e1;
        //                 B[x - 1][j + 2] = e2;
        //                 B[x - 1][j + 3] = e3;
        //                 B[x - 1][j + 4] = e4;
        //                 B[x - 1][j + 5] = e5;
        //                 B[x - 1][j + 6] = e6;
        //                 B[x - 1][j + 7] = e7;

        //                 e0 = A[x][j];
```

```
        //                 e1 = A[x][j + 1];
        //                 e2 = A[x][j + 2];
        //                 e3 = A[x][j + 3];
        //                 e4 = A[x][j + 4];
        //                 e5 = A[x][j + 5];
        //                 e6 = A[x][j + 6];
        //                 e7 = A[x][j + 7];
        //             }

        //             B[i + 7][j] = e0;
        //             B[i + 7][j + 1] = e1;
        //             B[i + 7][j + 2] = e2;
        //             B[i + 7][j + 3] = e3;
        //             B[i + 7][j + 4] = e4;
        //             B[i + 7][j + 5] = e5;
        //             B[i + 7][j + 6] = e6;
        //             B[i + 7][j + 7] = e7;

        //             x = B[i][j + 1]; B[i][j + 1] = B[j
+ 1][i]; B[j + 1][i] = x;
        //             x = B[i][j + 2]; B[i][j + 2] = B[j
+ 2][i]; B[j + 2][i] = x;
        //             x = B[i][j + 3]; B[i][j + 3] = B[j
+ 3][i]; B[j + 3][i] = x;
        //             x = B[i][j + 4]; B[i][j + 4] = B[j
+ 4][i]; B[j + 4][i] = x;
        //             x = B[i][j + 5]; B[i][j + 5] = B[j
+ 5][i]; B[j + 5][i] = x;
        //             x = B[i][j + 6]; B[i][j + 6] = B[j
+ 6][i]; B[j + 6][i] = x;
        //             x = B[i][j + 7]; B[i][j + 7] = B[j
+ 7][i]; B[j + 7][i] = x;

        //             x = B[i + 1][j + 2]; B[i + 1][j +
2] = B[j + 2][i + 1]; B[j + 2][i + 1] = x;
        //             x = B[i + 1][j + 3]; B[i + 1][j +
3] = B[j + 3][i + 1]; B[j + 3][i + 1] = x;
        //             x = B[i + 1][j + 4]; B[i + 1][j +
4] = B[j + 4][i + 1]; B[j + 4][i + 1] = x;
        //             x = B[i + 1][j + 5]; B[i + 1][j +
5] = B[j + 5][i + 1]; B[j + 5][i + 1] = x;
        //             x = B[i + 1][j + 6]; B[i + 1][j +
6] = B[j + 6][i + 1]; B[j + 6][i + 1] = x;
        //             x = B[i + 1][j + 7]; B[i + 1][j +
7] = B[j + 7][i + 1]; B[j + 7][i + 1] = x;

        //             x = B[i + 2][j + 3]; B[i + 2][j +
3] = B[j + 3][i + 2]; B[j + 3][i + 2] = x;
        //             x = B[i + 2][j + 4]; B[i + 2][j +
4] = B[j + 4][i + 2]; B[j + 4][i + 2] = x;
        //             x = B[i + 2][j + 5]; B[i + 2][j +
5] = B[j + 5][i + 2]; B[j + 5][i + 2] = x;
```

```
        //             x = B[i + 2][j + 6]; B[i + 2][j +
6] = B[j + 6][i + 2]; B[j + 6][i + 2] = x;
        //             x = B[i + 2][j + 7]; B[i + 2][j +
7] = B[j + 7][i + 2]; B[j + 7][i + 2] = x;

        //             x = B[i + 3][j + 4]; B[i + 3][j +
4] = B[j + 4][i + 3]; B[j + 4][i + 3] = x;
        //             x = B[i + 3][j + 5]; B[i + 3][j +
5] = B[j + 5][i + 3]; B[j + 5][i + 3] = x;
        //             x = B[i + 3][j + 6]; B[i + 3][j +
6] = B[j + 6][i + 3]; B[j + 6][i + 3] = x;
        //             x = B[i + 3][j + 7]; B[i + 3][j +
7] = B[j + 7][i + 3]; B[j + 7][i + 3] = x;

        //             x = B[i + 4][j + 5]; B[i + 4][j +
5] = B[j + 5][i + 4]; B[j + 5][i + 4] = x;
        //             x = B[i + 4][j + 6]; B[i + 4][j +
6] = B[j + 6][i + 4]; B[j + 6][i + 4] = x;
        //             x = B[i + 4][j + 7]; B[i + 4][j +
7] = B[j + 7][i + 4]; B[j + 7][i + 4] = x;

        //             x = B[i + 5][j + 6]; B[i + 5][j +
6] = B[j + 6][i + 5]; B[j + 6][i + 5] = x;
        //             x = B[i + 5][j + 7]; B[i + 5][j +
7] = B[j + 7][i + 5]; B[j + 7][i + 5] = x;

        //             x = B[i + 6][j + 7]; B[i + 6][j +
7] = B[j + 7][i + 6]; B[j + 7][i + 6] = x;

        //         } else {
        //             for(x = i; x < i + 8; x++)
        //                 for(y = j; y < j + 8; y++)
        //                     B[y][x] = A[x][y];
        //         }
        //     }
        // }

    }
    else if(M == 32) { //32x32 matrix
        /**
         * for any N and M both divided by 8
         */
        for(i = 0; i < 32; i += 8) {
            for(j = 0; j < 32; j += 8) {
                // diag: 11 * 4
                for(x = i; x < i + 4; x++) {
                    e0 = A[x][j];
                    e1 = A[x][j + 1];
                    e2 = A[x][j + 2];
                    e3 = A[x][j + 3];
                    e4 = A[x][j + 4];
                    e5 = A[x][j + 5];
                    e6 = A[x][j + 6];
                    e7 = A[x][j + 7];

                    B[j][x] = e0;
                    B[j + 1][x] = e1;
                    B[j + 2][x] = e2;
                    B[j + 3][x] = e3;
                    B[j][x + 4] = e4;
                    B[j + 1][x + 4] = e5;
                    B[j + 2][x + 4] = e6;
                    B[j + 3][x + 4] = e7;
                }

                // diag: 15 * 4
                for(x = j; x < j + 4; x++) {
                    e0 = A[i + 4][x];
                    e1 = A[i + 5][x];
                    e2 = A[i + 6][x];
                    e3 = A[i + 7][x];
                    e4 = B[x][i + 4];
                    e5 = B[x][i + 5];
                    e6 = B[x][i + 6];
                    e7 = B[x][i + 7];

                    B[x][i + 4] = e0;
                    B[x][i + 5] = e1;
                    B[x][i + 6] = e2;
                    B[x][i + 7] = e3;
                    B[x + 4][i] = e4;
                    B[x + 4][i + 1] = e5;
                    B[x + 4][i + 2] = e6;
                    B[x + 4][i + 3] = e7;
                }

                // diag: 9 * 4
                for(x = i + 4; x < i + 8; x++) {
                    e0 = A[x][j + 4];
                    e1 = A[x][j + 5];
                    e2 = A[x][j + 6];
                    e3 = A[x][j + 7];
                    B[j + 4][x] = e0;
                    B[j + 5][x] = e1;
                    B[j + 6][x] = e2;
                    B[j + 7][x] = e3;
                }
            }
        }
    } else if(M == 48) { // 32x48 matrix
        // for(i = 0; i < 32; i += 8) {
        //     for(j = 0; j < 48; j += 8) {
        //         // diag: 11 * 4
        //         for(x = i; x < i + 4; x++) {
```

```
//             e0 = A[x][j];
//             e1 = A[x][j + 1];
//             e2 = A[x][j + 2];
//             e3 = A[x][j + 3];
//             e4 = A[x][j + 4];
//             e5 = A[x][j + 5];
//             e6 = A[x][j + 6];
//             e7 = A[x][j + 7];

//             B[j][x] = e0;
//             B[j + 1][x] = e1;
//             B[j + 2][x] = e2;
//             B[j + 3][x] = e3;
//             B[j][x + 4] = e4;
//             B[j + 1][x + 4] = e5;
//             B[j + 2][x + 4] = e6;
//             B[j + 3][x + 4] = e7;
//         }

//         // diag: 15 * 4
//         for(x = j; x < j + 4; x++) {
//             e0 = A[i + 4][x];
//             e1 = A[i + 5][x];
//             e2 = A[i + 6][x];
//             e3 = A[i + 7][x];
//             e4 = B[x][i + 4];
//             e5 = B[x][i + 5];
//             e6 = B[x][i + 6];
//             e7 = B[x][i + 7];

//             B[x][i + 4] = e0;
//             B[x][i + 5] = e1;
//             B[x][i + 6] = e2;
//             B[x][i + 7] = e3;
//             B[x + 4][i] = e4;
//             B[x + 4][i + 1] = e5;
//             B[x + 4][i + 2] = e6;
//             B[x + 4][i + 3] = e7;
//         }

//         // diag: 9 * 4
//         for(x = i + 4; x < i + 8; x++) {
//             e0 = A[x][j + 4];
//             e1 = A[x][j + 5];
//             e2 = A[x][j + 6];
//             e3 = A[x][j + 7];
//             B[j + 4][x] = e0;
//             B[j + 5][x] = e1;
//             B[j + 6][x] = e2;
//             B[j + 7][x] = e3;
//         }
//     }
```

```
        // }

        /**
         * for violent blocking
         */
        // for(i = 0; i < N; i += 17) {
        //     for(j = 0; j < M; j += 17) {
        //         for(x = 0; x < i + 17 && x < N; x++) {
        //             for(y = 0; y < j + 17 && y < M;
y++) {
        //                 B[y][x] = A[x][y];
        //             }
        //         }
        //     }
        // }

        /**
         * for violent but first copy blocking 8x8
         */
        for(i = 0; i < N / 8 * 8; i += 8)
            for(j = 0; j < M / 8 * 8; j += 8) {
                for(x = 0; x < 8; x++) {
                    e0 = A[i + x][j];
                    e1 = A[i + x][j + 1];
                    e2 = A[i + x][j + 2];
                    e3 = A[i + x][j + 3];
                    e4 = A[i + x][j + 4];
                    e5 = A[i + x][j + 5];
                    e6 = A[i + x][j + 6];
                    e7 = A[i + x][j + 7];

                    B[j + x][i] = e0;
                    B[j + x][i + 1] = e1;
                    B[j + x][i + 2] = e2;
                    B[j + x][i + 3] = e3;
                    B[j + x][i + 4] = e4;
                    B[j + x][i + 5] = e5;
                    B[j + x][i + 6] = e6;
                    B[j + x][i + 7] = e7;
                }

                for(x = 0; x < 8; x++) {
                    for(y = 0; y < x; y++) {
                        e1 = B[j + x][i + y];
                        B[j + x][i + y] = B[j + y][i + x];
                        B[j + y][i + x] = e1;
                    }
                }
            }
        for (i = N / 8 * 8; i < N; ++i)
            for (j = M / 8 * 8; j < M; ++j)
            {
                e1 = A[i][j];
                B[j][i] = e1;
            }
        for (i = 0; i < N; ++i)
            for (j = M / 8 * 8; j < M; ++j)
            {
                e1 = A[i][j];
                B[j][i] = e1;
            }
        for (i = N / 8 * 8; i < N; ++i)
            for (j = 0; j < M; ++j)
            {
                e1 = A[i][j];
                B[j][i] = e1;
            }

    }
}
```

for 32x48 excellent

```
/*
 * trans.c - Matrix transpose B = A^T
 *
 * Your transpose function must have a prototype of the
form:
 * void trans(int M, int N, int A[N][M], int B[M][N]);
 *
 * The transpose function is evaluated by counting the
number of misses
 * on a 512B direct mapped cache with a block size of 32
bytes.
 */
#include "cachelab.h"
#include <stdio.h>

/*
 * transpose_submit - This is the solution transpose
function that you
 *     will be graded on for Part B of the assignment. Do
not change
 *     the description string "Transpose submission", as
the driver
 *     searches for that string to identify the transpose
function to
 *     be graded.
 */

void transpose_submit(int M, int N, int A[N][M], int B[M]
[N])
{
```

```
/*
 * Please define your local variables here.
 * You are allowed to define at most 12 local variables.
 */
int i, j, k;
//int tmp0, tmp1, tmp2, tmp3, tmp4, tmp5, tmp6, tmp7;

if (M == 48)
{ // 48x32 mxn
  for (k = 0; k < 48; k += 16)
  {
    if (k == 16)
    {
      for (i = 0; i < 4; i += 1)
      {
        for (j = 0; j < 8; j += 1)
        {
          B[k + i][8 + j] = A[i * 2 + 1][k + j];
        }
      }
      for (i = 0; i < 8; i += 1)
      {
        for (j = 0; j < 8; j += 1)
        {
          if(j % 2 == 1)
          {
            B[k + i][j] = B[k + j / 2][i + 8];
          }
          else
          {
            B[k + i][j] = A[j][k + i];
          }
        }
      }

      for (i = 0; i < 8; i += 1)
      {
          for (j = 0; j < 8; j += 1)
          {
            B[k + i][8 + j] = A[8 + j][k + i];
          }
      }

      for (i = 0; i < 4; i += 1)
      {
        for (j = 0; j < 8; j += 1)
        {
          B[k + 8 + i][j] = A[8 + i * 2 + 1][k + 8 + j];
        }
      }
      for (i = 0; i < 8; i += 1)
      {
```

```
     for (j = 0; j < 8; j += 1)
     {
      if (j % 2 == 1)
      {
        B[k + 8 + i][8 + j] = B[k + 8 + j / 2][i];
      }
      else
      {
        B[k + 8 + i][8 + j] = A[8 + j][k + 8 + i];
      }
     }
    }

    for (i = 0; i < 8; i += 1)
    {
     for (j = 0; j < 8; j += 1)
     {
      B[k + 8 + i][j] = A[j][k + 8 + i];
     }
    }

    for(i = 0; i < 4; i += 1)
    {
     for (j = 0; j < 8; j += 1)
     {
      B[k + i][24 + j] = A[16 + i * 2][k + j];
     }
    }
    for (i = 0; i < 8; i += 1)
    {
     for (j = 0; j < 8; j += 1)
     {
      if (j % 2 == 0)
      {
        B[k + i][16 + j] = B[k + j / 2][16 + i +
8];
      }else
      {
       B[k + i][16 + j] = A[16 + j][k + i];
      }
     }
    }

    for (i = 0; i < 8; i += 1)
    {
     for (j = 0; j < 8; j += 1)
     {
      B[k + i][24 + j] = A[24 + j][k + i];
     }
    }

    for (i = 0; i < 4; i += 1)
    {
     for (j = 0; j < 8; j += 1)
     {
      B[k + 8 + i][16 + j] = A[24 + i * 2][k + 8 +
j];
     }
    }
    for (i = 0; i < 8; i += 1)
    {
     for (j = 0; j < 8; j += 1)
     {
      if (j % 2 == 0)
      {
       B[k + 8 + i][24 + j] = B[k + 8 + j / 2][24 +
i - 8];
      }else
      {
       B[k + 8 + i][24 + j] = A[24 + j][k + 8 + i];
      }
     }
    }

    for (i = 0; i < 8; i += 1)
    {
     for (j = 0; j < 8; j +=1)
     {
      B[k + 8 + i][16 + j] = A[16 + j][k + 8 + i];
     }
    }
   }
   else
   {
    for (i = 0; i < 4; i += 1)
    {
     for (j = 0; j < 8; j += 1)
     {
      B[k + i][8 + j] = A[i * 2][k + j];
     }
    }
    for (i = 0; i < 8; i += 1)
    {
     for (j = 0; j < 8; j += 1)
     {
      if(j % 2 == 0)
      {
       B[k + i][j] = B[k + j / 2][i + 8];
      }
      else
      {
       B[k + i][j] = A[j][k + i];
      }
     }
```

```
}

for (i = 0; i < 8; i += 1)
{
    for (j = 0; j < 8; j += 1)
    {
      B[k + i][8 + j] = A[8 + j][k + i];
    }
}

for (i = 0; i < 4; i += 1)
{
  for (j = 0; j < 8; j += 1)
  {
    B[k + 8 + i][j] = A[8 + i * 2][k + 8 + j];
  }
}
for (i = 0; i < 8; i += 1)
{
  for (j = 0; j < 8; j += 1)
  {
    if (j % 2 == 0)
    {
        B[k + 8 + i][8 + j] = B[k + 8 + j / 2][i];
    }
    else
    {
        B[k + 8 + i][8 + j] = A[8 + j][k + 8 + i];
    }
  }
}

for (i = 0; i < 8; i += 1)
{
  for (j = 0; j < 8; j += 1)
  {
    B[k + 8 + i][j] = A[j][k + 8 + i];
  }
}

for(i = 0; i < 4; i += 1)
{
  for (j = 0; j < 8; j += 1)
  {
    B[k + i][24 + j] = A[16 + i * 2 + 1][k + j];
  }
}
for (i = 0; i < 8; i += 1)
{
  for (j = 0; j < 8; j += 1)
  {
    if (j % 2 == 1)
```

```
            {
                B[k + i][16 + j] = B[k + j / 2][16 + i +
8];
            }else
            {
              B[k + i][16 + j] = A[16 + j][k + i];
            }
          }
        }

        for (i = 0; i < 8; i += 1)
        {
          for (j = 0; j < 8; j += 1)
          {
            B[k + i][24 + j] = A[24 + j][k + i];
          }
        }

        for (i = 0; i < 4; i += 1)
        {
          for (j = 0; j < 8; j += 1)
          {
            B[k + 8 + i][16 + j] = A[24 + i * 2 + 1][k + 8
+ j];
          }
        }
        for (i = 0; i < 8; i += 1)
        {
          for (j = 0; j < 8; j += 1)
          {
            if (j % 2 == 1)
            {
              B[k + 8 + i][24 + j] = B[k + 8 + j / 2][24 +
i - 8];
            }else
            {
              B[k + 8 + i][24 + j] = A[24 + j][k + 8 + i];
            }
          }
        }

        for (i = 0; i < 8; i += 1)
        {
          for (j = 0; j < 8; j +=1)
          {
            B[k + 8 + i][16 + j] = A[16 + j][k + 8 + i];
          }
        }
      }

    }
  }
```

```
 }
```